Latent Fingerprint SDK Test (Phase II)

API Specification

Introduction

The Latent Fingerprint SDK Test provides a means of determining core search performance of latent-fingerprint matchers. This document specifies all SDK interfaces and functionality as well as the data formats used for this test.

There will be minimal human involvement during the actual execution of the test. A small amount of human assistance will probably be required to prepare the data. All such assistance will be provided indirectly by NIST, and may include:

a) Crop and orient certain latents.

b) Provide a *region-of-interest*.

c) Provide latent experts for examining potential *consolidations*.

Those wishing to submit software for Latent Fingerprint SDK testing shall be required to provide NIST with an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface) specified in this document.

1 Fingerprint Image Data

1.1 *Format*

The SDK must be capable of processing fingerprint images supplied to the SDK in uncompressed raw 8-bit (one byte per pixel) grayscale format. The image data shall appear to be the result of a scanning of a conventional inked impression of a fingerprint. Figure 1 illustrates the recording order for the scanned image. The origin is the upper left corner of the image. The x-coordinate (horizontal) position shall increase positively from the origin to the right side of the image. The y-coordinate (vertical) position shall increase positively from the origin to the bottom of the image.

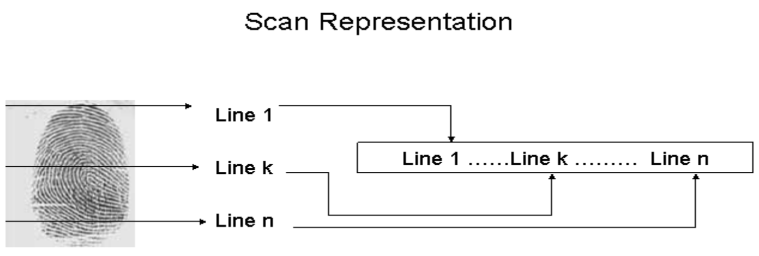

Figure 1 Order of scanned lines

Raw 8-bit grayscale images are canonically encoded. The minimum value that will be assigned to a "black" pixel is zero. The maximum value that will be assigned to a "white" pixel is 255. Intermediate gray levels will have assigned values of 1- 254. The pixels are stored left to right, top to bottom, with one 8-bit byte per pixel. The number of bytes in an image is equal to its height multiplied by its width as measured in pixels; there is no header. The image height and width in pixels will be supplied to the SDK as supplemental information.

1.2 *Resolution, Dimensions and Orientation*

The latent fingerprint images will employ either 500 or 1000 ppi resolution (both horizontal and vertical). All background fingerprint images will employ 500 ppi resolution (both horizontal and vertical). The precise resolution for each individual image will be specified to the SDK via the API.

All fingerprint images used for the background will vary from 150 to 1000 pixels in both width and height dimensions. All latent images at 500 ppi will vary from 150 to 1000 pixels in both width and height. All latent images at 1000 ppi will vary from 150 to 2000 pixels in both width and height. The precise dimensions of each individual image will be specified to the SDK via the API.

All latent fingerprint images used for Phase II testing may vary in orientation over the full angular range (0 to 359). The estimated orientation and range of uncertainty of individual latent prints may be specified to the SDK via the API. Otherwise, the orientation is specified as “upright” +-180 degrees. No information will be specified to the SDK regarding the orientation of background fingerprint images.

No information regarding the distribution of fingerprint image resolution, dimensions, or orientation within the Phase II dataset is provided in this document.

2 Test Interface Description

Participants shall submit an SDK which provides the interfaces defined in section 2.3. Section 2.2 defines the interfaces to functions provided by NIST for use by the SDK. Sections 2.1 and 2.4 specify the declaration of constants, error codes, data-types and functions used by both.

2.1 *Declarations*

The following are declarations of data types and functions used in the Latent Fingerprint SDK testing interface:

```
//////////////////////////////////////////////////////
// Declarations of constants                        //
//////////////////////////////////////////////////////

// Impression type codes
#define IMPTYPE_LP      0     // Live-scan plain
#define IMPTYPE_LR      1     // Live-scan rolled
#define IMPTYPE_NP      2     // Nonlive-scan plain
#define IMPTYPE_NR      3     // Nonlive-scan rolled

// Finger position codes
#define FINGPOS_UK      0     // Unknown finger
#define FINGPOS_RT      1     // Right thumb
#define FINGPOS_RI      2     // Right index finger
#define FINGPOS_RM      3     // Right middle finger
#define FINGPOS_RR      4     // Right ring finger
#define FINGPOS_RL      5     // Right little finger
#define FINGPOS_LT      6     // Left thumb
```

```
#define FINGPOS_LI      7     // Left index finger
#define FINGPOS_LM      8     // Left middle finger
#define FINGPOS_LR      9     // Left ring finger
#define FINGPOS_LL      10    // Left little finger

////////////////////////////////////////////////////////////////
// Declarations for the NIST provided library functions       //
////////////////////////////////////////////////////////////////

// Structure to hold a single fingerprint record (image+metadata)
struct finger_record
{
      BYTE    impression_type;
      UINT16  resolution;     // Image resolution in pixels/cm
      BYTE    finger_position;
      UINT16  height;         // Image height in pixels
      UINT16  width;          // Image width in pixels
      BYTE    *image_data;    // 8-bit grayscale image data
};
typedef struct finger_record   FINGER_REC;

// Extracts 10 fingerprint records from a ten-print (AN2K) file
INT32 extract_image_data(const char *tenprint_filename,
      FINGER_REC **finger_recs);

// De-allocates the memory holding 10 fingerprint records
void free_image_data(FINGER_REC *finger_recs);

////////////////////////////////////////////////////////////////
// Declarations for the SDK provided library functions        //
////////////////////////////////////////////////////////////////

// Structure to hold zero or more candidates returned in a search
struct candidate {
      UINT32 background_index;
      BYTE   finger_position;
      DOUBLE similarity_score;
      BYTE   probability;
      UINT16 num_matching_minutiae;
      BYTE   candidate_quality;
}
typedef struct candidate CANDIDATE;

// Structure to hold list of candidates returned by SDK
struct candidate_list
{
      UINT32    num_entries;
      UINT16    num_latent_minutiae;
      BYTE      latent_quality;
      CANDIDATE *list;
};
typedef struct candidate_list   CANDIDATE_LIST;

// Enrolls the entire set of background images
INT32 enroll_background(const INT32 num_recs,
      const char **filenames, const char *enrollment_dir,
      char *error_msg);
```

```
// Selects the current background for latent image searching
INT32 set_background(const char *enrollment_dir);

// Enrolls the latent image
INT32 enroll_latent(const FINGER_REC *latent_finger,
      const BYTE *roi_mask, const UINT16 orientation,
      const BYTE offset, BYTE  *enrolled_latent,
      INT32 *enroll_length);

// Searches for the latent image in the background
INT32 image_search(const BYTE *enrolled_latent,
      CANDIDATE_LIST *candidates, char *error_msg);
```

2.2 *NIST Provided Functions*

2.2.1 Extract Image Data

```
INT32
extract_image_data(const char  *tenprint_filename,
                   FINGER_REC  **finger_recs);
```

Description

This function extracts ten fingerprint image records from a single (AN2K formatted) ten-print record file. The caller shall pass ***tenprint_filename*** as a pointer to the fully qualified pathname of an AN2K formatted ten-print record file, and ***finger_recs*** as the address of a pointer of type `FINGER_REC` (see 2.1 above).

Upon return ***finger_recs*** will contain a pointer to an array of ten `FINGER_REC` structures ordered by finger position from 1 (right thumb) to 10 (left little finger). For any fingers that are missing from the original ten-print record file, the *image_data* field in the respective `FINGER_REC` will be a NULL pointer.

Example

```
// Example of processing a ten-print record
FINGER_REC *finger_recs;
INT32 status=extract_image_data(“image00205.an2k”, &finger_recs);
if (status == 0) {
      for (i=0;i<10;i++) {
            if (finger_recs[i].image_data != NULL)
                  process_valid_finger(finger_recs[i]);
            else
                  process_missing_finger(finger_recs[i]);
      }
      free_image_data(finger_recs); // see 2.2.2 below
}
```

Parameters

`tenprint_filename` **(input)**: A pointer to a ten-print record filename.
`finger_recs` **(output)**: The address of a `FINGER_REC` pointer.

Return Value

This function returns *zero* on success or a documented *non-zero* error code otherwise.

2.2.2 Free Image Data

```
void
free_image_data(FINGER_REC *finger_recs);
```

Description

De-allocates all memory used by the array of `FINGER_REC` structures specified by ***finger_recs*** which was allocated during a call to `extract_image_data().`

Parameters

`finger_recs` **(input)**: A pointer to an array of `FINGER_REC` structures.

Return Value

None.

2.3 *SDK Provided Functions*

2.3.1 Enroll Background

```
INT32
enroll_background(const INT32  num_recs,
                  const char   **filenames,
                  const char   *enrollment_dir,
                  char         *error_msg);
```

Description

This function performs the conversion of all background 10-print records into a proprietary dataset. No format is prescribed for this data, but it could be a set of proprietary templates. Pre-computation of background data avoids reprocessing of the original images upon subsequent calls to `image_search()`.

The SDK shall use the function `extract_image_data()` (see 2.2.1 above) provided by NIST to extract the raw grayscale image and metadata from each 10-print record file specified in the ***filenames*** array. Note that each call to `extract_image_data()` allocates memory to hold the extracted image and metadata, so this memory should be de-allocated using the NIST provided `free_image_data()` (see 2.2.2 above) function when no longer needed.

The format of the filenames pointed to by the ***filenames*** array will be canonical Unix style pathnames using forward slash directory separators (e.g. "/mnt1/xyz/foo-22/image00205.an2k").

All data produced by the SDK shall be stored exclusively to the directory specified by ***enrollment_dir***. The contents of this directory are at the discretion of the vendor.

Non-fatal error conditions shall be tolerated and shall not result in pre-mature halting (i.e. non-completion of background enrollment). These error conditions include missing fingers in 10-print records, and failure-to-enroll (FTE) any portion of a 10-print record. If any of the above non-fatal error conditions are encountered, the SDK may optionally return a documented non-zero warning code (after completing background enrollment), though this is not required.

Upon entry the ***error_msg*** parameter will point to a pre-allocated and pre-zeroized string buffer of length 513 bytes (512 + 1 for the *NULL* terminator) which the SDK may use for outputting detailed information regarding fatal errors which have occurred (signaled by a non-zero return code). This may be useful for debugging any problems that might occur after the SDK is received by NIST. For example, if the enrollment process encounters a fatal or non-fatal error during processing of a specific background ten-print record file, the SDK could output an error message including the ten-print record filename to ***error_msg*** and return a documented non-zero error or warning code respectively.

Note 1: The order of the ten-print record file names in *filenames* defines (implicitly) the indexing scheme that shall be used henceforth for recording the ten-print record indices of all candidates returned by *`image_search()`*. The index of the first ten-print record is 1.

Note 2: During subsequent calls to *`image_search()`* the SDK is permitted to access the original background images. To support this access, the path information supplied by *filenames* regarding the original background images should be stored in the proprietary background set in *enrollment_dir*.

Parameters

`num_recs` **(input)**: The number of ten-print records to enroll.

`filenames` **(input)**: Array of pointers to all ten-print record filenames.

`enrollment_dir` **(input)**: The directory used to store enrollment data output.

`error_msg` **(output)**: Pointer to a detailed error message string.

Return Value

This function returns *zero* on success or a documented *non-zero* error code otherwise.

2.3.2 Set Background

```
INT32
set_background(const char  *enrollment_dir);
```

Description

This function selects the background that shall be used by all subsequent calls to `image_search()`. The directory specified by ***enrollment_dir*** shall contain the enrollment data produced by a prior call to `enroll_background()`.

Parameters

`enrollment_dir` **(input)**: The directory to be used by `image_search()`.

Return Value

This function returns *zero* on success or a documented *non-zero* error code otherwise.

2.3.3 Enroll Latent

```
INT32
enroll_latent(const FINGER_REC *latent_finger,
              const BYTE       *roi_mask,
              const UINT16     orientation,
              const BYTE       offset,
              BYTE             *enrolled_latent,
            INT32            *enroll_length);
```

Description

This function enrolls the latent image pointed to by ***latent_finger***, and writes the enrollment data to the memory location pointed by ***enrolled_latent***. The latent image itself shall be in "raw" uncompressed 8-bit grayscale format. No format is prescribed for the enrollment data.

The fields ***latent_finger->width*** and ***latent_finger->height*** specify the width and height of the latent image in pixels. The field ***latent_finger->resolution*** specifies the horizontal and vertical resolution of the latent image in pixels per centimeter (e.g. 500 pixels per inch is specified as 197 ppcm ; 1000 ppi is specified as 394 ppcm). The fields ***latent_finger->impression_type*** and ***latent_finger->finger_position*** will always be set equal to 0.

The function may be optionally supplied with a "region of interest" in the form of an image mask. In cases where no "region of interest" information is provided, the input ***roi_mask*** parameter shall be a *NULL* pointer. Otherwise, ***roi_mask*** shall point to a "raw" uncompressed raw 8-bit grayscale image with the same dimensions as the latent fingerprint image. The region (or regions) of interest in the latent fingerprint image are identified by the corresponding *x,y* locations in the ***roi_mask*** having non-zero pixels.

The ***orientation*** parameter specifies the estimated angle of the fingerprint in degrees (0 to 359). The ***offset*** (0 to 180) specifies the offset (+ or -) in degrees around this angle of allowable variance. Taken together these values inform the SDK as to fingerprint image's range of rotational variance which may be useful to the matching algorithm. The angle is expressed in standard mathematical format, with zero degrees to the right and angles increasing in the counterclockwise direction. Thus "upright" fingerprint images are said to have an orientation of 90 degrees. For example, if ***orientation*** and ***offset*** are specified as 75 and 5 respectively, the fingerprint image is estimated to have an orientation between 70 and 80 degrees. The ***offset*** 180 will only be used in conjunction with an ***orientation*** of 90 to convey complete uncertainty as to the fingerprint's orientation, and in that case full rotational variance (0 to 359) shall be assumed.

The memory for ***enrolled_latent*** is allocated prior to the call (i.e. by the application linked with the SDK) as a pre-zeroized 10 megabyte array.

Upon return from this function, ***enroll_length*** shall be set by the SDK to the length (in bytes) of the enrollment data stored in ***enrolled_latent***. The memory for ***enroll_length*** is allocated by the caller prior to calling this function.

Failure-to-enroll a latent shall result in a non-zero return code and upon return from this function the enrollment data written to ***enrolled_latent*** shall contain non-zero length data defined by the SDK as representing "null enrollment data." This "null enrollment data" shall be usable in subsequent searches for the corresponding latent, and result in the output of a candidate list with all entries set to 0.

Note that during the call to this function the directory containing the current background and its contents are read-only.

Parameters

`latent_finger` **(input)**: Pointer to a latent fingerprint image record.

`roi_mask` **(input)**: Pointer to optional image mask identifying ROI(s).

`orientation` **(input)**: The estimated orientation (in degrees) of the latent fingerprint.

`offset` **(input)**: The range of variance (in degrees) + or - the `orientation.`

`enrolled_latent` **(output)**: Pointer to memory block receiving the enrollment data.

`enroll_length` **(output)**: Pointer to length of `enrolled_latent` in bytes.

Return Value

This function returns *zero* on success or a documented *non-zero* error code on failure.

2.3.4 Image Search

```
INT32
image_search(const BYTE        *enrolled_latent,
             CANDIDATE_LIST    *candidates,
             char              *error_msg);
```

Description

This function searches the current background (as selected by `set_background()`) for zero or more candidates matching the input ***enrolled_latent*** parameter. The selection of features on which to match is entirely at the discretion of the SDK. Note that during the call to this function the directory containing the current background and its contents are read-only.

When this function is called, the ***candidates*** parameter will point to a pre-initialized `CANDIDATE_LIST` (see 2.1 above) with ***candidates->num_entries*** set equal to *M*, the number of background records (*N*) multiplied by 10 (i.e. *M* = *N* x 10), and ***candidates->list*** pointing to a pre-allocated *M* length array of (pre-zeroized) `CANDIDATE` structures.

During execution of this function the SDK shall modify the `CANDIDATE_LIST` structure such that ***candidates->num_entries*** is set equal to the number of candidates found (*S*), and the first *S* members of the array specified by ***candidates->list*** contain all candidate information. In other words, the first *S* structures of type `CANDIDATE` (see 2.1 above) pointed to by ***candidates->list*** shall contain the original background record file index, finger position, similarity score, and probability (range 0 to 100) for each candidate found by the search. For Phase I and II testing, the number of candidates found, S, shall equal 50 (and M will always be greater than 50). Additionally, the `CANDIDATE` structures in ***candidates->list*** shall be stored in decreasing order of *similarity_score*. Note that before returning from this function the SDK <u>must</u> set ***candidates->num_entries*** equal to 50, even if less than 50 candidates are actually written to ***candidates->list***. In the event that less than 50 candidates are actually written to ***candidates->list***, the pre-zeroized CANDIDATE structures in the array will effectively provide "padding" (with NULL candidates) to the required length of 50.

The *background_index* field for each `CANDIDATE` shall be set equal to the relative offset of the original ten-print record file processed by `enroll_background()`. The *finger_position* for each `CANDIDATE` shall be set equal to the finger position information extracted from its associated ten-print record file. And the *similarity_score* for each `CANDIDATE` shall be set to a value greater than or equal to 0 which represents the similarity of the input latent finger image to the respective candidate finger image in the

background. Note that any background fingerprint images not represented by an entry in ***candidates->list*** shall be implicitly assigned a similarity score equal to 0.
The *probability* field for each CANDIDATE shall be set equal to the probability (0-100) that the candidate is a "likely hit."

Non-fatal error conditions shall be tolerated and shall not result in pre-mature halting (i.e. non-completion of the search). These error conditions include encountering "gaps" in the background resulting from prior failure-to-enroll (FTE) events, and searching with an ***enrolled_latent*** containing "null enrollment data." In the latter case, the candidate list returned shall have all entries set to 0. If any of the above non-fatal error conditions are encountered, the SDK may optionally return a documented non-zero warning code (after completing the search), though this is not required.

Duplicate `CANDIDATE` entries or entries whose *background_index* field values are out of range (i.e. not between 1 and the N inclusive) shall not be accepted.

Upon entry the ***error_msg*** parameter will point to a pre-allocated and pre-zeroized string buffer of length 513 bytes (512 + 1 for the *NULL* terminator) that the SDK may use for outputting detailed information regarding fatal errors which have occurred (signaled by a non-zero return code). This may be useful for debugging any problems that might occur after the SDK is received by NIST.

Optionally, the quality of the latent print, the number of minutiae found in the latent print, the number of latent minutiae matching each candidate print, and the quality of the each candidate print may be returned (respectively) via the fields ***candidates->latent_quality*** **,** ***candidates->num_latent_minutiae***, ***candidate->num_matching_minutiae***, and ***candidate->candidate_quality***. If image quality values are supplied for either the latent or candidate print, the table below indicates the required range of values and their associated meanings:

Image Quality Value	Description
20	Poor
40	Fair
60	Good
80	Very Good
100	Excellent

Note 1: Matcher architectures in which "advanced matchers" are selectively invoked (depending upon initial screening results for the latent) are allowed. The SDK might decide to invoke (call) computationally intensive matchers only for those comparisons which show initial good results. However, the SDK must decide if the additional features (if any) used by these "advanced matchers" will be written to persistent storage during the call to `enroll_background()`.

Note 2: Since it may not be possible to keep all background images in memory, it might be necessary for the software to repeatedly retrieve the data from disk, and this extra fetch time will be included in the execution time measurement.

Note 3: The candidate list shall only depend on the inputs to this function and the currently selected background (not on any previous results from this function). Thus, identical inputs and background shall produce identical candidate lists independent of all prior calls to this function.

Parameters

`enrolled_latent` **(input)**: Pointer to the latent image's enrollment data.

`candidates` **(input/output)**: A list of candidates matching the latent fingerprint image.

`error_msg` **(output)**: Pointer to a detailed error message string.

Return Value

This function returns *zero* on success or a documented *non-zero* error code on failure.

2.4 *Error Codes and Handling*

The participant shall provide documentation of all (non-zero) error or warning return codes (see section 3.3, Documentation).

The application should include error/exception handling so that in the case of a fatal error, the return code is still provided to the calling application.

All messages which convey errors, warnings or other information shall be suppressed. Information supplemental to the documented error codes returned by the SDK shall be conveyed via the ***error_msg*** parameter (see 2.3 above) only.

At minimum the following return codes shall be used.

Return code	Function	Explanation
0	All	Success
-1	extract_image_data()	unable to open file
-2	extract_image_data()	Incorrect file format
-3	extract_image_data()	error parsing ten-print file
-4	extract_image_data()	error decompressing image
-5	extract_image_data()	insufficient memory error
-6	extract_image_data()	unspecified error
100	enroll_background()	enrollment directory not found
101	enroll_background()	error extracting image(s) from ten print
102	enroll_background()	error writing enrollment data
103	enroll_background()	insufficient memory error
200	set_background()	enrollment directory not found
300	enroll_latent()	image size not supported
301	enroll_latent()	image resolution not supported
302	enroll_latent()	insufficient features found in latent
400	image_search()	enrollment directory not set
401	image_search()	insufficient memory available for search
402	image_search()	unable to access original ten-print record

3 Software and Documentation

3.1 *SDK Library and Platform Requirements*

Individual SDKs shall not include multiple "modes" of operation, or algorithm variations which require explicit activation by the calling application. If participants wish to separately compare the performance of such features, they must submit separate SDKs. Note that this requirement does not preclude implementation of internally (i.e. autonomously) selected modes or algorithm variations within a single SDK. Only such features requiring external selection by the calling application are forbidden.

Participants shall provide NIST with binary code only (i.e. no source code) − supporting files such as header (“.h”) files notwithstanding. It is preferred that the SDK be submitted in the form of a single static library file (ie. “.LIB” for Windows or “.a” for Linux). However, dynamic/shared library files are permitted.

If dynamic/shared library files are submitted, it is preferred that the API interface specified by this document be implemented in a single “core” library file with the base filename ‘liblatent’ (for example, ‘liblatent.dll’ for Windows or ‘liblatent.so’ for Linux). Additional dynamic/shared library files may be submitted that support this “core” library file (i.e. the “core” library file may have dependencies implemented in these other libraries).

Note that dependencies on external dynamic/shared libraries such as compiler-specific development environment libraries are discouraged. If absolutely necessary, external libraries must be provided to NIST upon prior approval by the Test Liaison.

The SDK will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the library code provided shall not use any interactive functions such as graphical user interface (GUI) calls, or any other calls which require terminal interaction (e.g. calls to “standard input” or “standard output”).

The use of multi-threading by the SDK is encouraged as the NIST test platform includes dual-processor support. The SDK need not be “thread safe” as the NIST test driver itself is single threaded. If multi-threading is utilized by the SDK is shall be documented.

NIST will link the provided library file(s) to a C language test driver application (developed by NIST) using the GCC compiler (*for Windows platforms Cygwin/GCC version 3.3.1 will be used; for Linux platforms GCC version 2.96 and GNU ld 2.11.90.0.8 will be used. All GCC compilers use Libc 6*). For example,

```
gcc  –o latenttest  latenttest.c  -L. –llatent
```

Participants are required to provide their library in a format that is linkable using GCC with the NIST test driver, which is compiled with GCC. All compilation and testing will be performed on x86 platforms running either Windows 2000 Professional SP4 (or higher) or Linux (kernel 2.4.7-10 or higher) dependent upon the operating system requirements of the SDK. Thus, participants are strongly advised to verify library-level compatibility with GCC (on an equivalent platform) prior to submitting their software to NIST to avoid linkage problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.).

3.2 *Installation and Usage*

The SDK must install easily (i.e. one installation step with no participant interaction required) to be tested, and shall be executable on any number of machines without requiring additional machine-specific license control procedures or activation.

The SDK’s usage shall be unlimited. No usage controls or limits based on licenses, execution date/time, number of executions, etc. shall be enforced by the SDK.

It is requested that the SDK be installable using simple file copy methods, and not require the use of a separate installation program. Contact the Test Liaison for prior approval if an installation program is absolutely necessary.

3.3 *Documentation*

Complete documentation of the SDK shall be provided, and shall detail any additional functionality or behavior beyond what is specified in this document. The documentation must define all error and warning codes.

Multi-threading behavior by an SDK shall be documented.

3.4 *Speed Requirement*

All times given assume the use of a 2.8GHz Pentium IV equivalent or faster processor. Time will be measured as “wall clock” elapsed time.

The average time to enroll a single background ten-print record shall take no more than 150 seconds (15 sec/image).

The average time to enroll a single latent image shall take no more than 600 seconds.

The average time to search a single background ten-print record shall take no more than 0.25 seconds (0.025 sec/image).